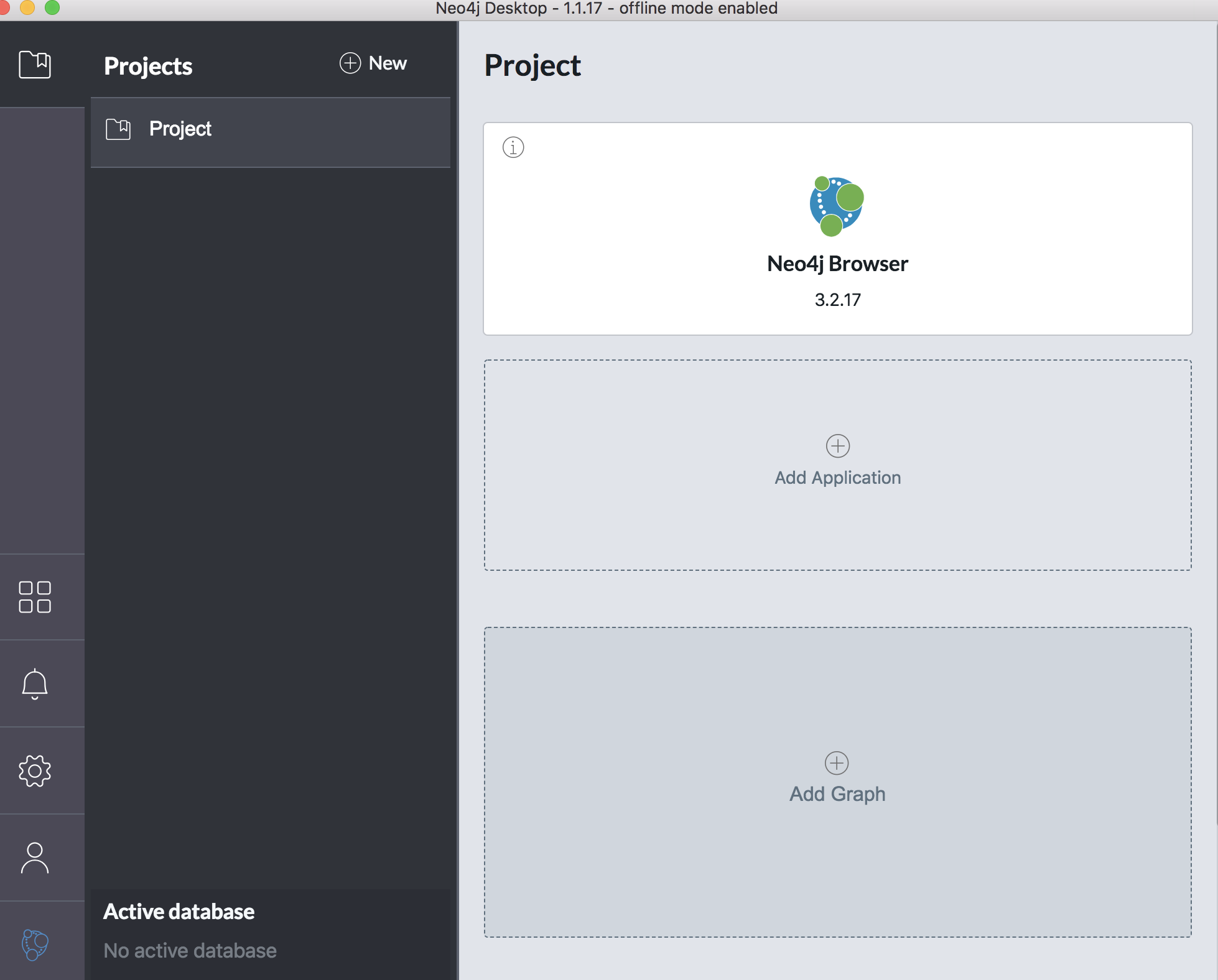
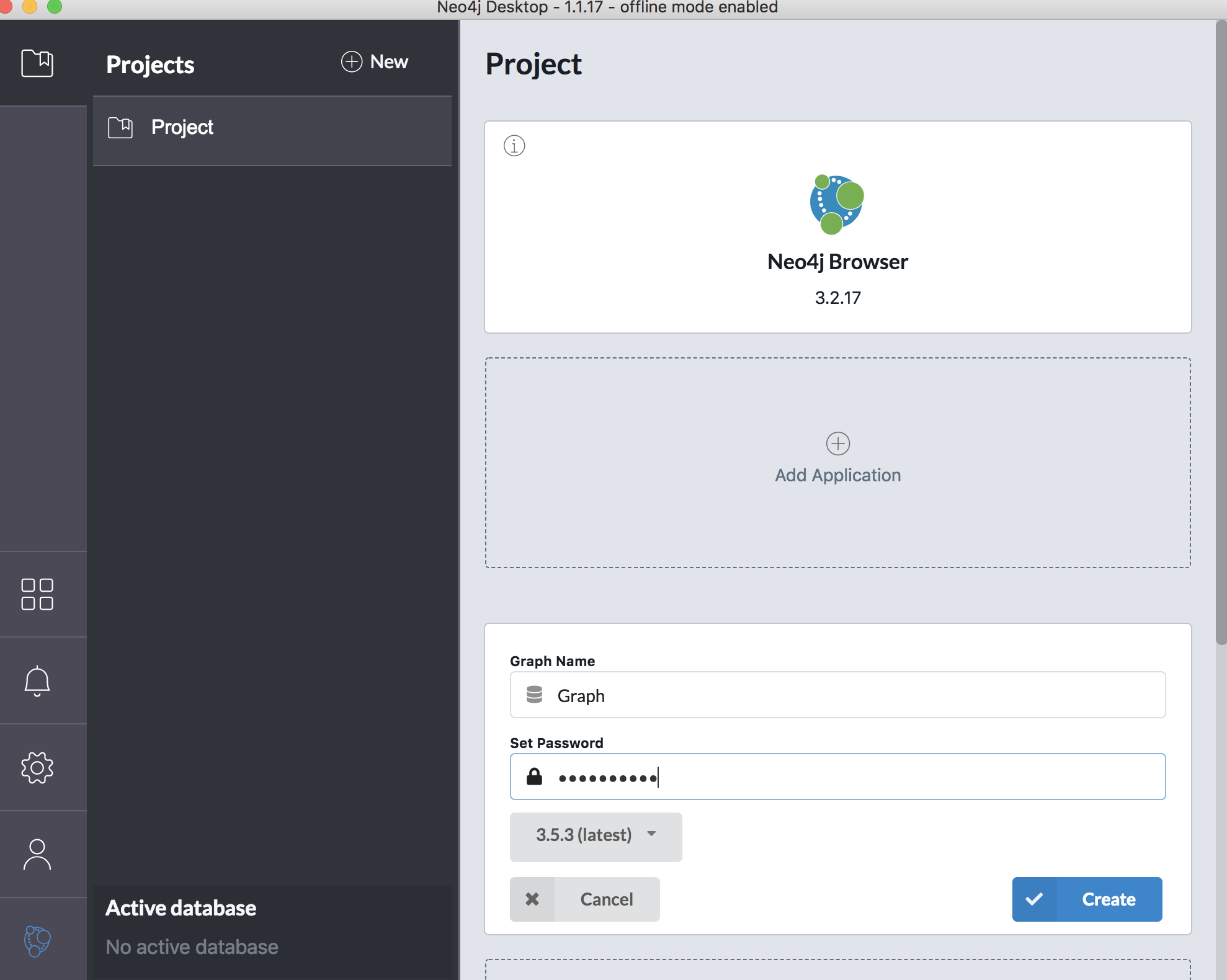
**Neo4j Week 1 Lab**

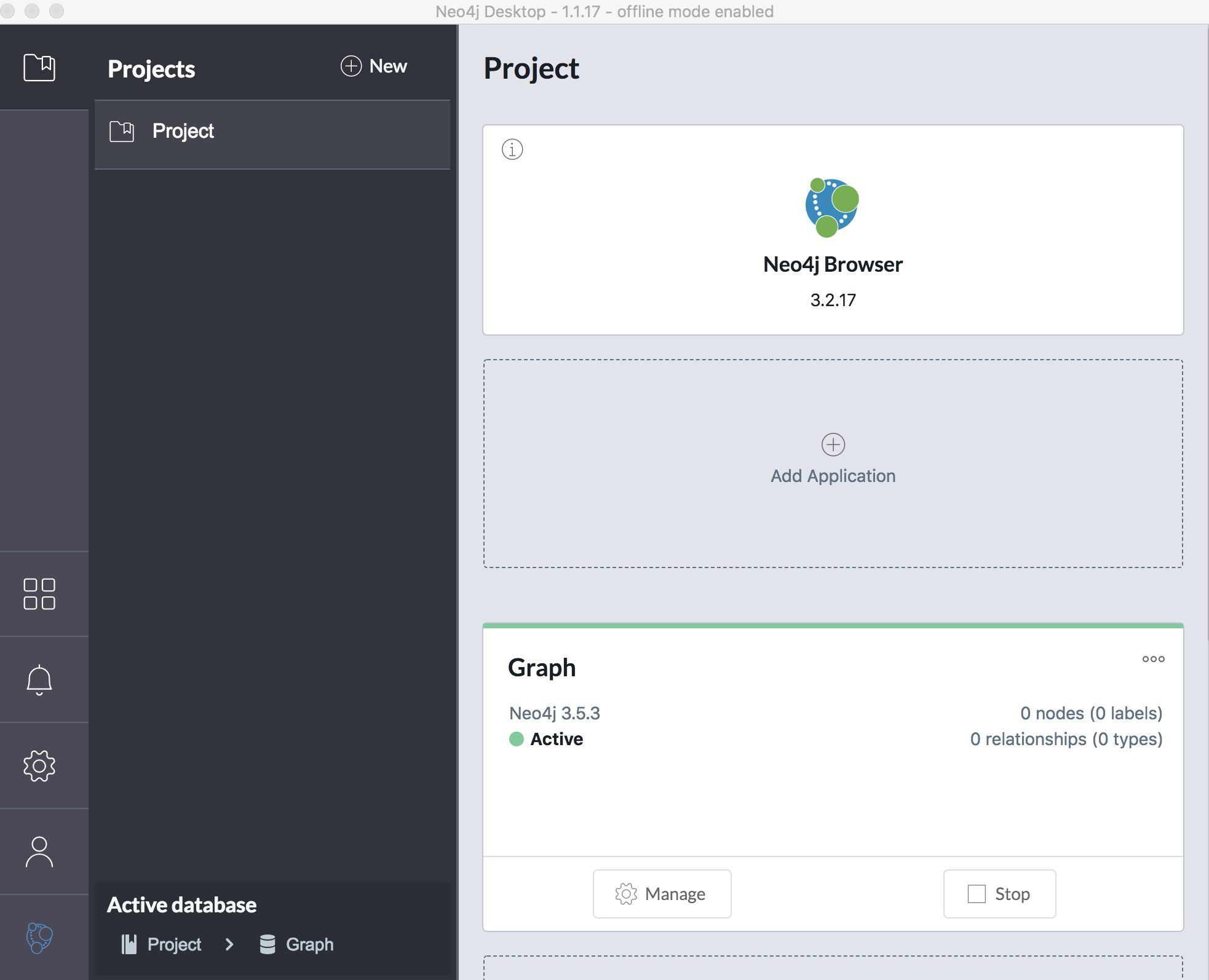
**Part I – Setting up your Neo4j environment**

1. Download Neo4j Desktop: <https://neo4j.com/download/>
2. In a project, create a local graph database 🡪 click Add Graph 🡪 Give it a password that you can remember 🡪 Click Create

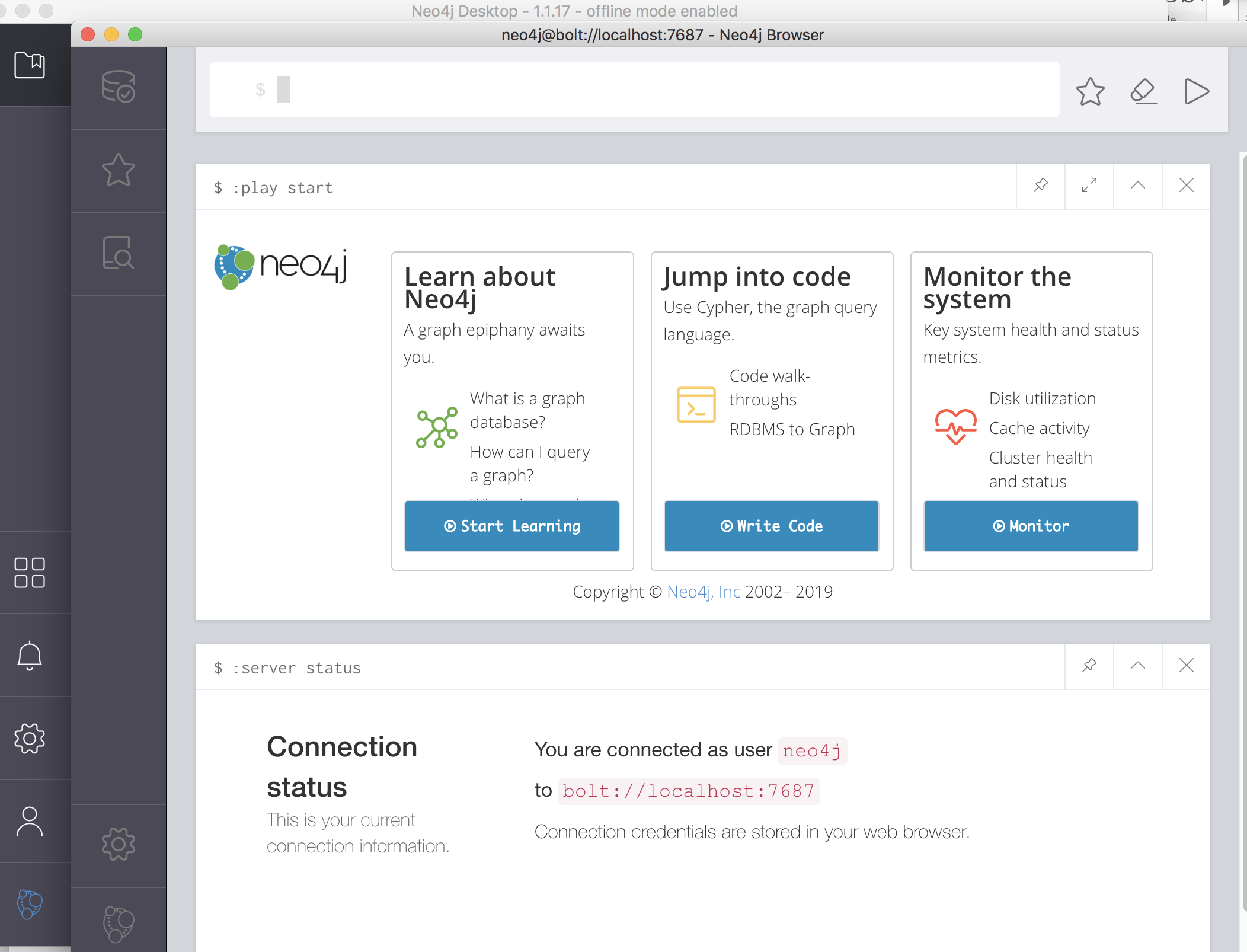




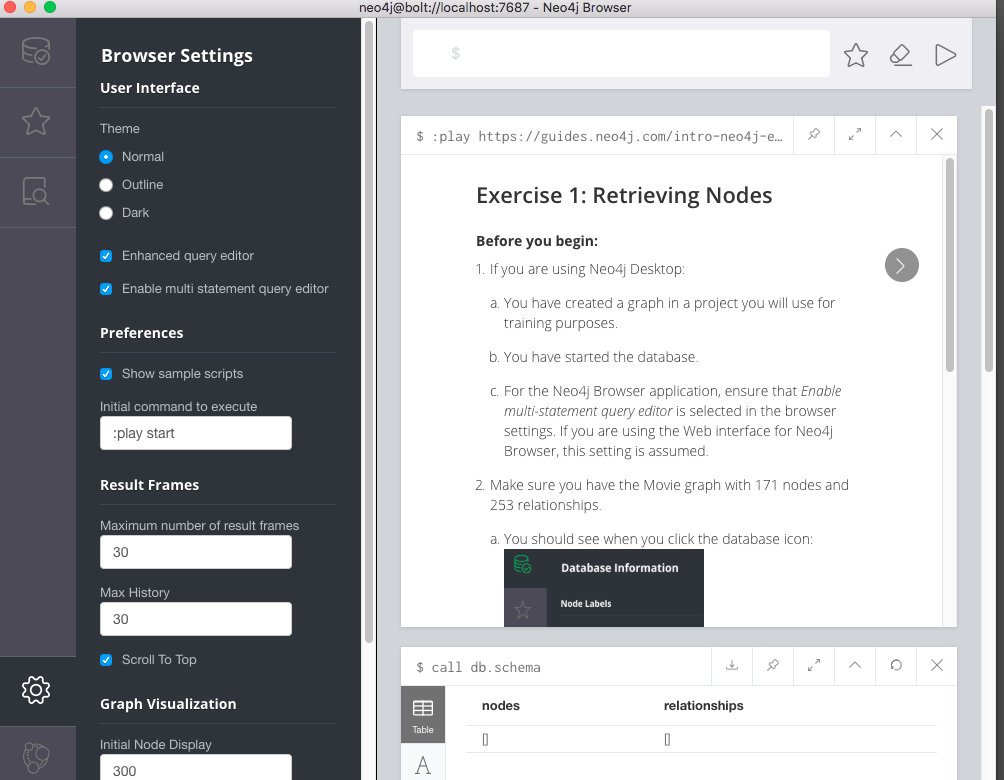
1. Start the database



1. Click on the Neo4j Browser application



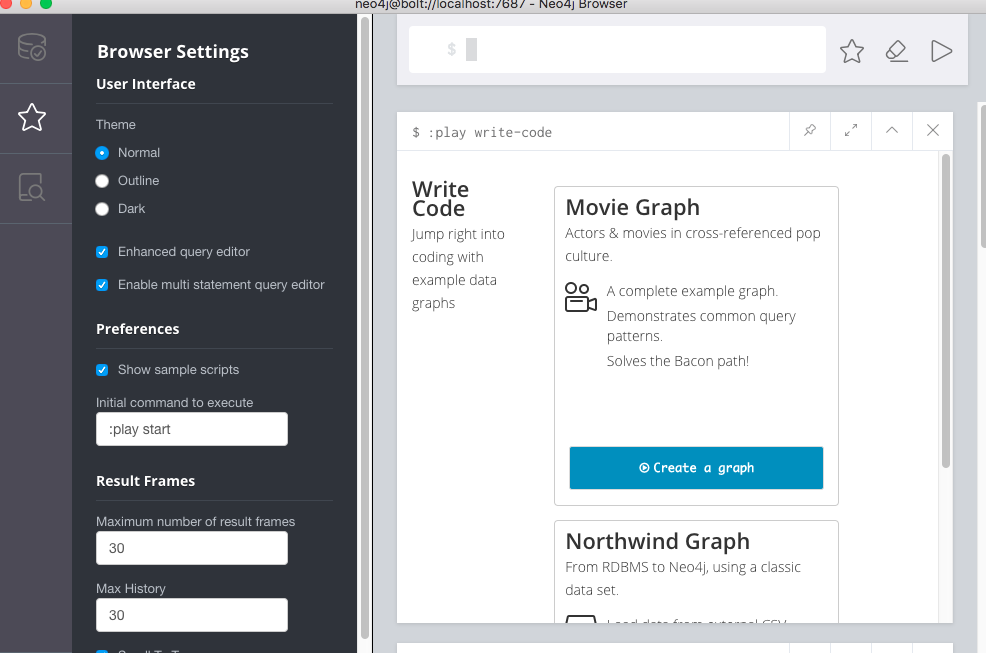
1. Make sure that **Enable multi-statement query editor** is enabled in browser settings:

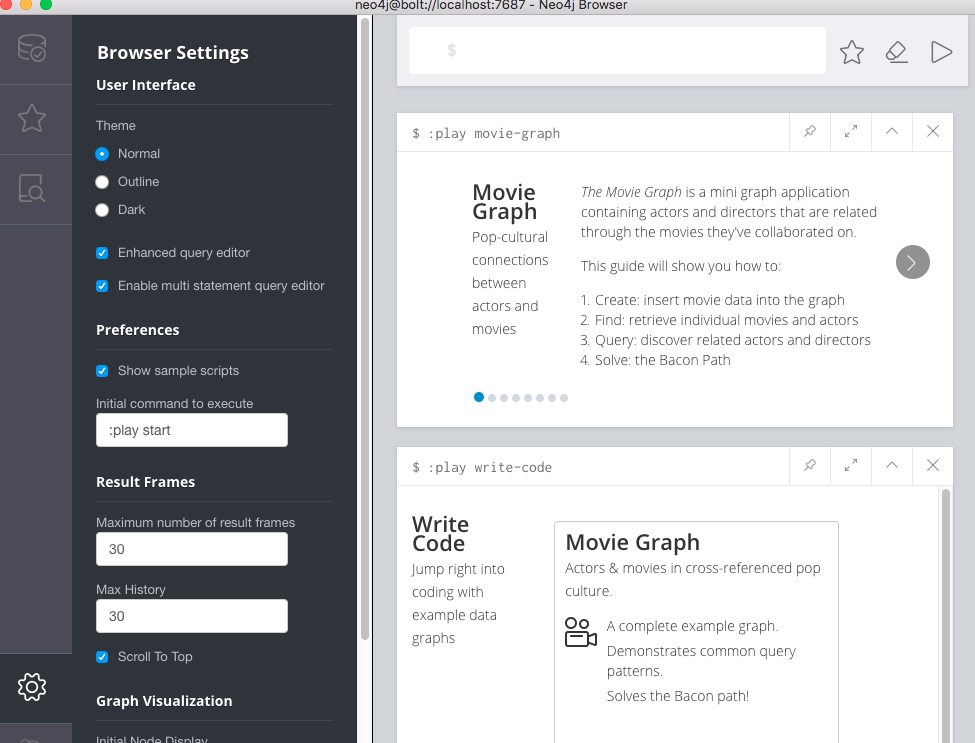


1. Create Movie database that you will use in this lab by typing the following into Neo4j browser:

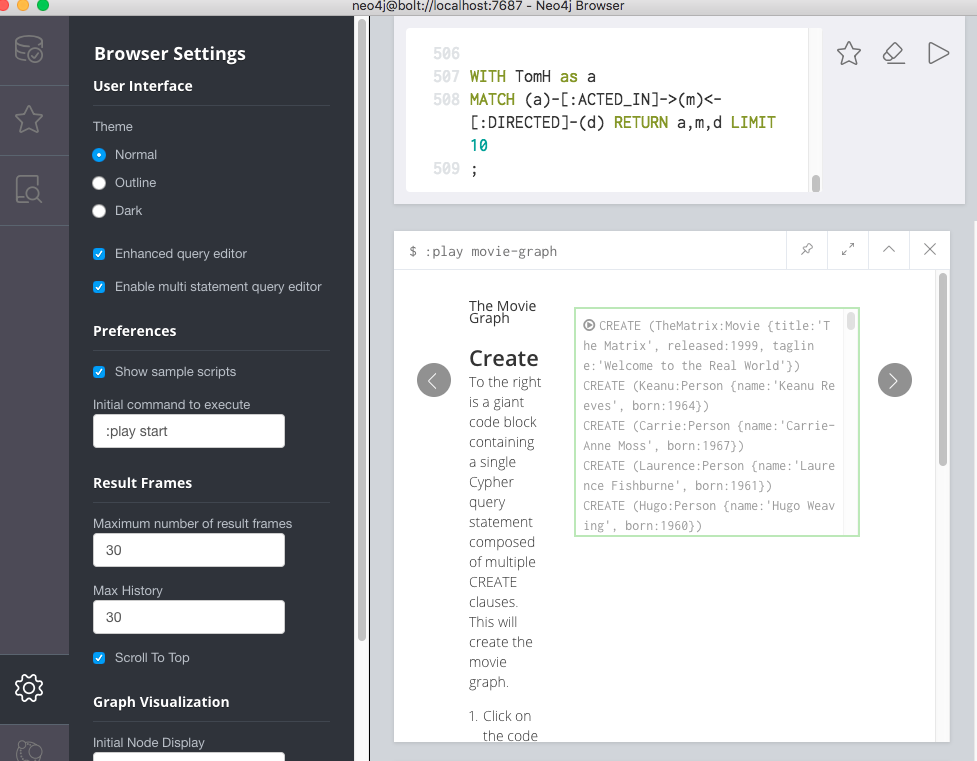
**:play write-code**

1. Click **Create a graph** and step through the guide to create it

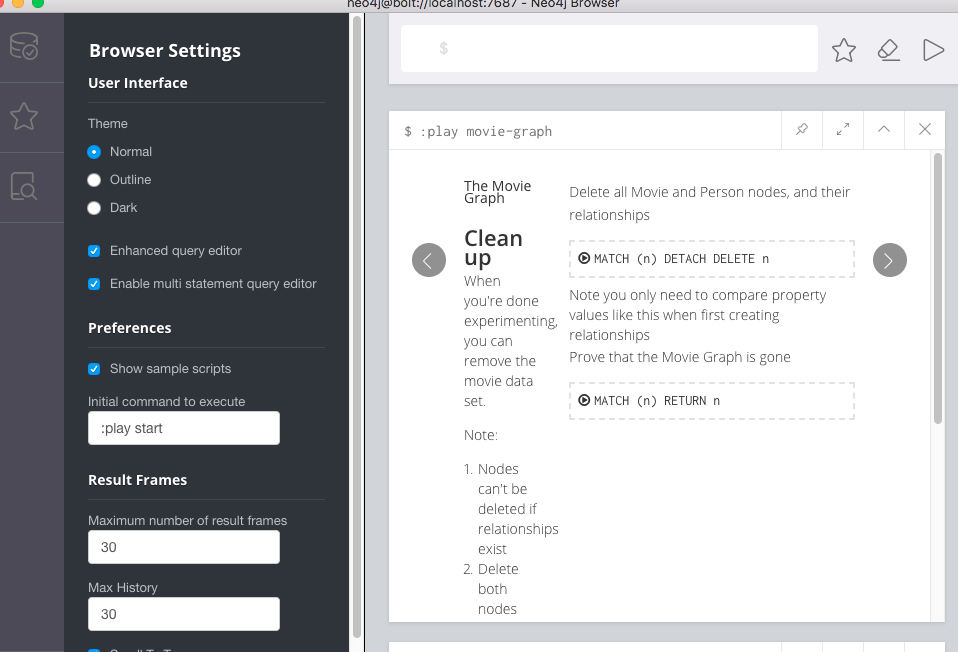
****



Click on the statement and it will get copy/pasted automatically into the browser. All you have to do is hit play button. DO NOT CLEAN UP (don’t execute DELETE statements):



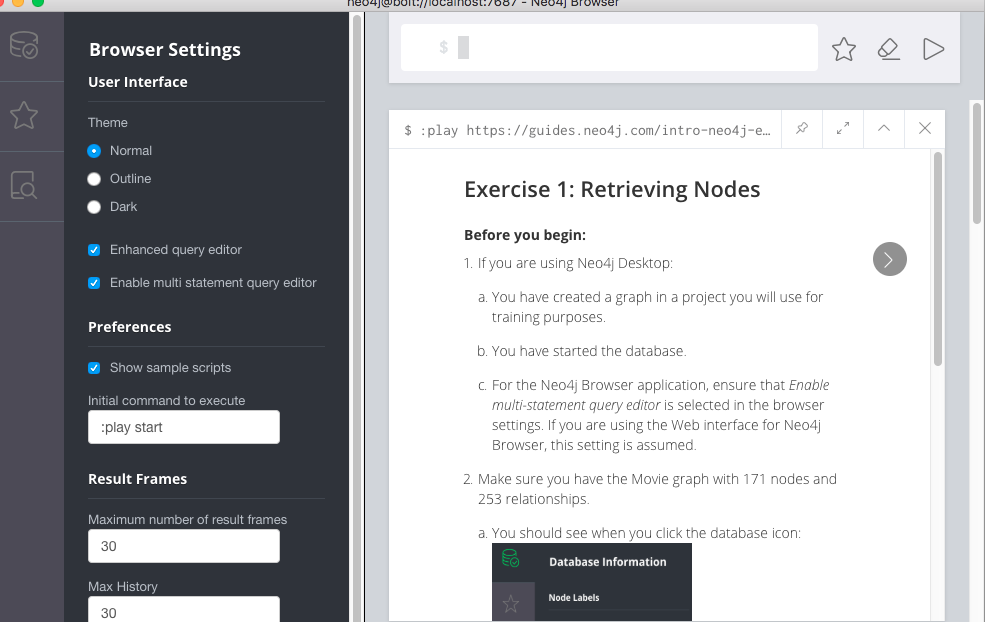
**DO NOT EXECUTE DELETE STATEMENTS, AS WE NEED MOVIE DATABASE FOR THIS LAB:**

****

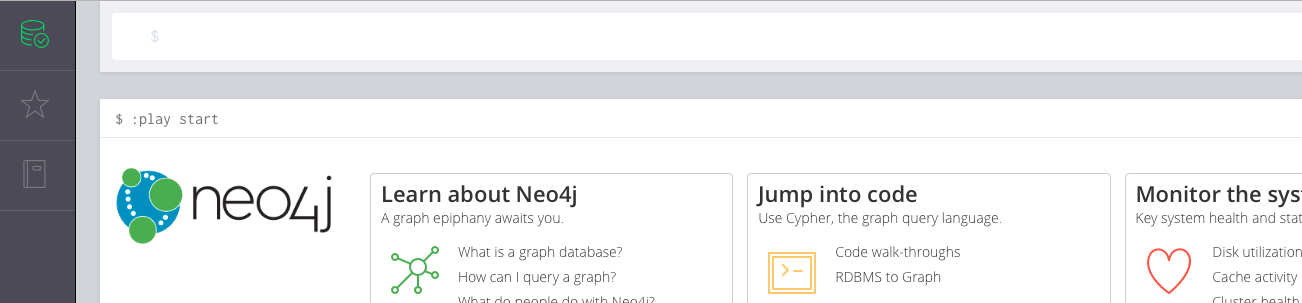
**Part II– Working with Cypher**

1. Inside Neo4j Browser type the following and follow instructions for **Exercise 1:**

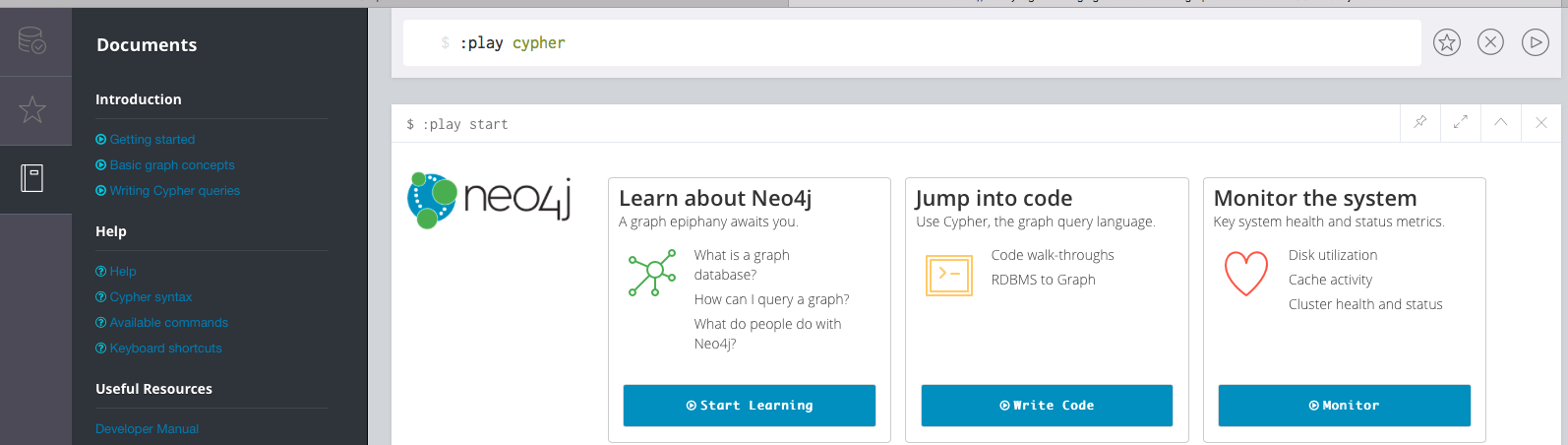
**:play https://guides.neo4j.com/intro-neo4j-exercises/01.html**



1. Continue to Exercise 2, step through the exercise steps. Don’t continue to Exercise 3
2. Let’s further explore Cypher language. We will create small local graph. Navigate to the Documents icon on the left:



Click **Writing Cypher queries 🡪 Play cypher**

****

**Part III – Writing more Cypher Queries**

1. **Write a query to explore what kind of nodes exist in our graph database.**

**match** (n)

**return** labels(n) **as** labels, keys(n) **as** keys, **count**(\*) **as** total

**order** **by** total **desc**;

**What this query does:**

1. The keyword match is followed by the pattern (n) that matches any node and assigns it to the variable n. The function call labels(n) returns a list of all labels associated with the node n
2. The function call keys(n) returns a list of all of the names associated with properties — these act much like column names in SQL.
3. count(\*) is an aggregate function, so the query implicitly groups by labels and keys.
4. **Write a query to extract information about the relationships in the database:**

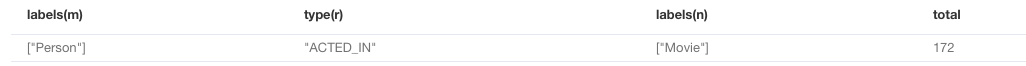
**match** (m)-[r]->(n)

**return** labels(m), **type**(r), labels(n), **count**(\*) **as** total

**order** **by** total **desc**;

**What this query does:**

1. The pattern (m)-[r]->(n) matches two nodes (m and n) and a relationship (r) from m to n (all relationships in Neo4j have a direction).
2. So you can read it like an arrow m --> n where the relationship r is enclosed in square brackets.
3. The function type(r) returns the type of the relationship r.
4. This tells us, for example, that there are 172 relationships of type ACTED\_IN between nodes with label Person and nodes of label Movie.



1. Write a query using the pattern () to match any node:

**match** ()-[r]->()

**return** **type**(r) **as** **type**, keys(r) **as** keys, **count**(\*) **as** total

**order** **by** **type**;

**What this query does:**

1. Each relationship between two nodes can also have a set of properties.
2. Look at the list of properties ACTED\_IN. We have modeled the ACTED\_IN relationship as having properties DIRECTED, PRODUCED, WROTE, REVIEWED and FOLLOWS. Note that not all instances of this relationship type have all of these properties. This is how Neo4j deals with the absence of data: whereas SQL has a special NULL value that indicates there is no value, Neo4j simply leaves out a property that has no associated value, so it doesn't appear in keys().
3. **Write a query to get all nodes and relationships between nodes, naming the columns**

**MATCH** (n)-[r]->(m) **RETURN** **n as node\_a, r as relates\_to, m as node\_b;**

1. **Write a query to get all nodes in a graph:**

**MATCH**(x)**RETURN** x

1. **Write a query to get a specific node within all of the nodes**

**MATCH**(person:Person)

**WHERE** person.name = “Hugo Weaving”

**RETURN** person;

1. **Write a query to get nodes of certain category (all data)**

**match** (a:Person)**return** **a**

1. **Write a query to name a relationship and return its type**

**MATCH** (actor)-[r]-> ()

**RETURN** actor.name, type(r);

1. **Write a query to match a relationship type**

**MATCH** (actor)-[:ACTED\_IN]-> (movie)

**RETURN** actor, movie;

1. **Write a query to get nodes for a specific property:**

**match** (a:Person) where n.name=”Ron Howard” **return** **a**

1. **Write a query to match all nodes:**

**match** (n) **return** **n**

1. **Match nodes according to relationships (undirected = bidirectional):**

**match** (a)-[:ACTED\_IN]-(b) **return** **a,b**

1. **Match nodes according to relationships (directed):**

**match** (a)-[:FOLLOWS]->(b) **return** **a,b**

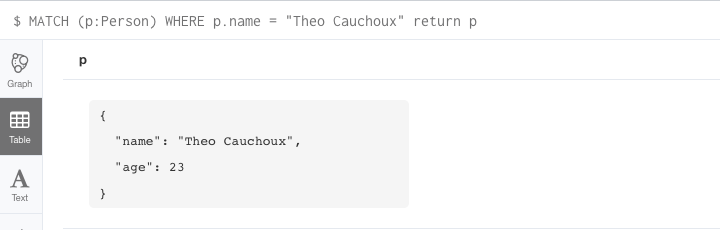
1. **Create a new node**

**CREATE** (a:Person {name: “Theo Cauchoux”}) **RETURN** **a**

1. **Update a specific property of a node:**

**MATCH** (p:Person) WHERE p.name = “Theo Cauchoux” **SET p.age = 23**

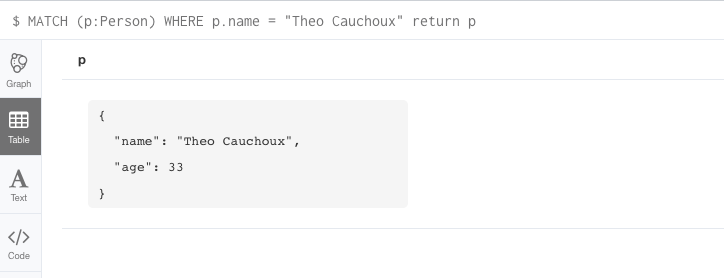
**Observe your update:**

****

1. **Replace all properties of a node:**

**MATCH** (p:Person) WHERE p.name = “Theo Cauchoux” **SET p.age = 33**

**Observe your replace:**

****

1. **Create a node with the name “Mystic River” and label “Movie”**

**CREATE** (:Movie {title:”Mystic River”, released:1993})

1. **Add a property (tagline) to a node (Mystic River)**

**MATCH** (movie:Movie)

**WHERE** movie.title= “Mystic River”

**SET** movie.tagline= “We bury our sins here Dave. We wash them clean”

**RETURN** movie;

1. **Create a relationship between two nodes (Kevin Bacon in Mystic River)**

**MATCH** (movie:Movie),(actor:Person)

**WHERE** movie.title= “Mystic River” AND actor.name= “Kevin Bacon”

**MERGE** (actor)-[role:ACTED\_IN]->(movie) ON CREATE SET role.roles=[“Sean”]

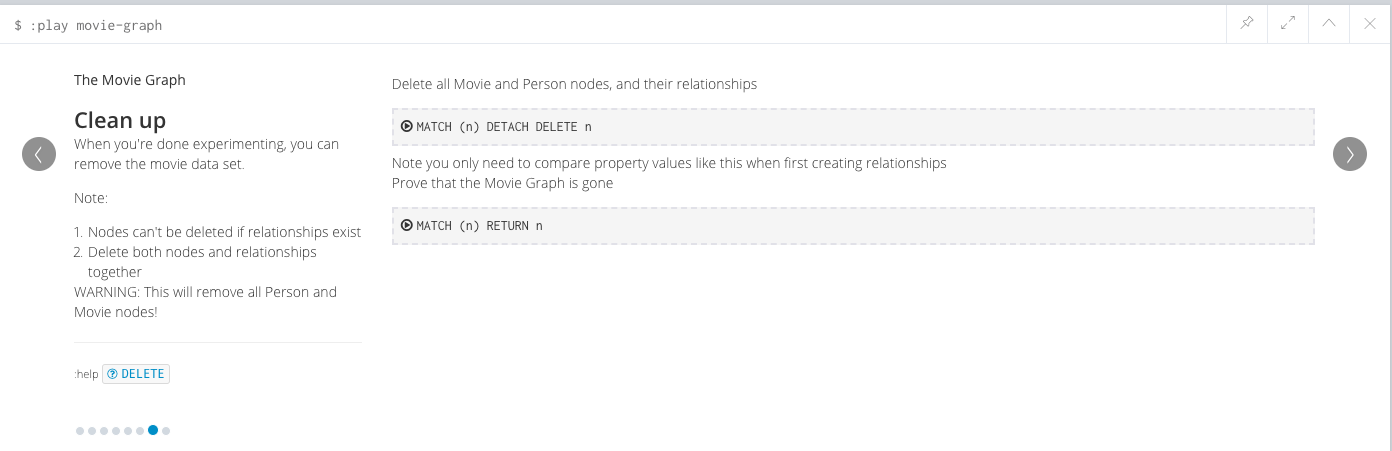
1. **Delete all nodes with the title Mystic River**

**MATCH** (movie:Movie)

**WHERE** movie.title= “Mystic River”

**DETACH DELETE movie;**

1. **Now run clean up:**

****

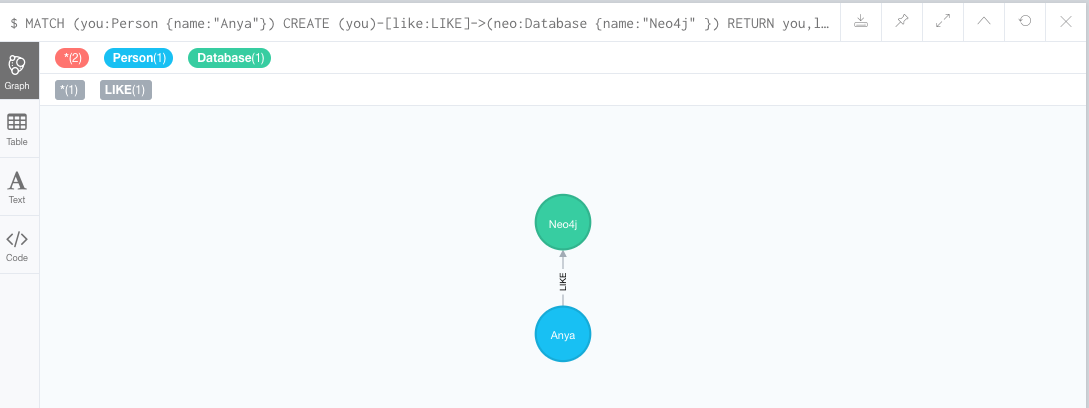
**Part IV – Create a Record For Yourself**

1. **Create node:**

**CREATE (you:Person {name:"YOUR\_NAME"}) RETURN you**

1. **Add new relationship to a node:**

**MATCH (you:Person {name:"You"}) CREATE (you)-[like:LIKE]->(neo:Database {name:"Neo4j" }) RETURN you,like,neo**

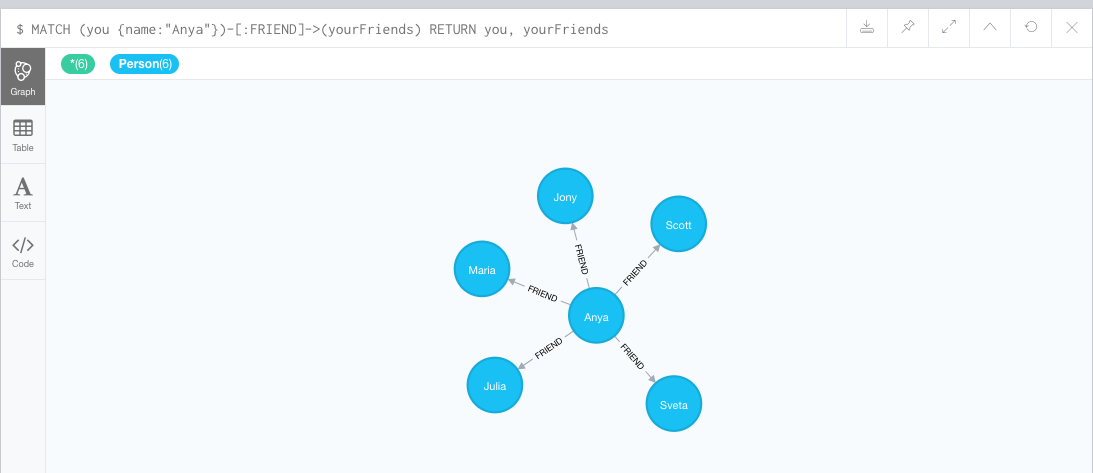


1. **Create your friends (**FOREACH lets you execute updates for each element in the list**:**

**MATCH (you:Person {name:"Anya"}) FOREACH (name in ["Maria","Julia","Scott","Jony","Sveta"] | CREATE (you)-[:FRIEND]->(:Person {name:name}))**

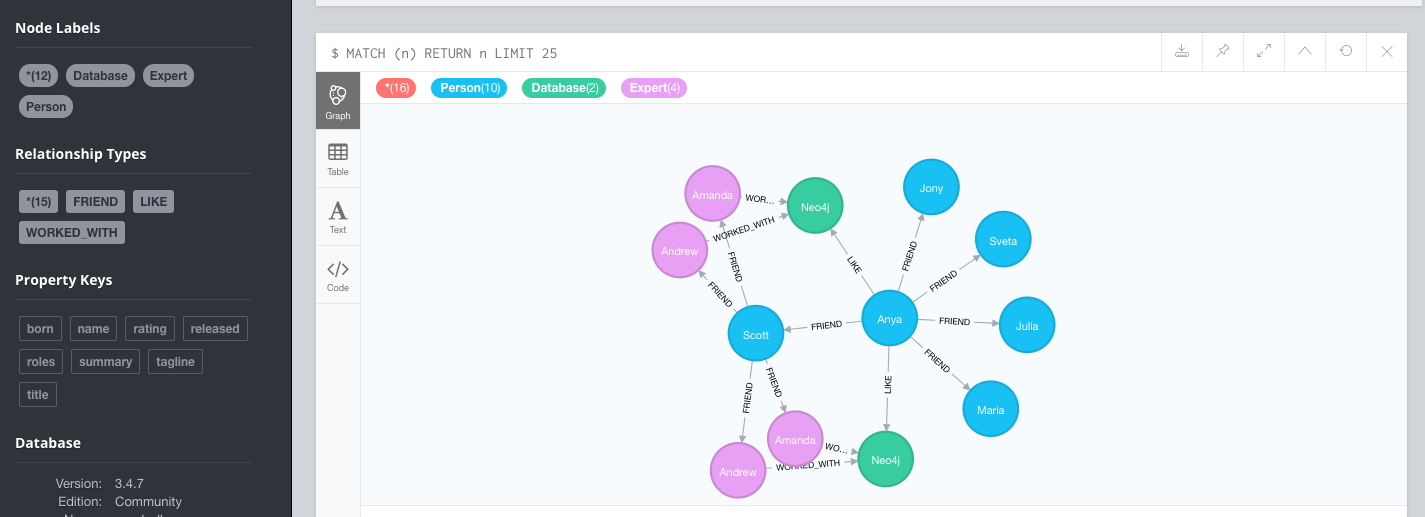
1. **Find all of your friends:**

**MATCH (you {name:"Anya"})-[:FRIEND]->(yourFriends) RETURN you, yourFriends**



1. **Create second degree friends and skills (**click on number of nodes to explore)**:**

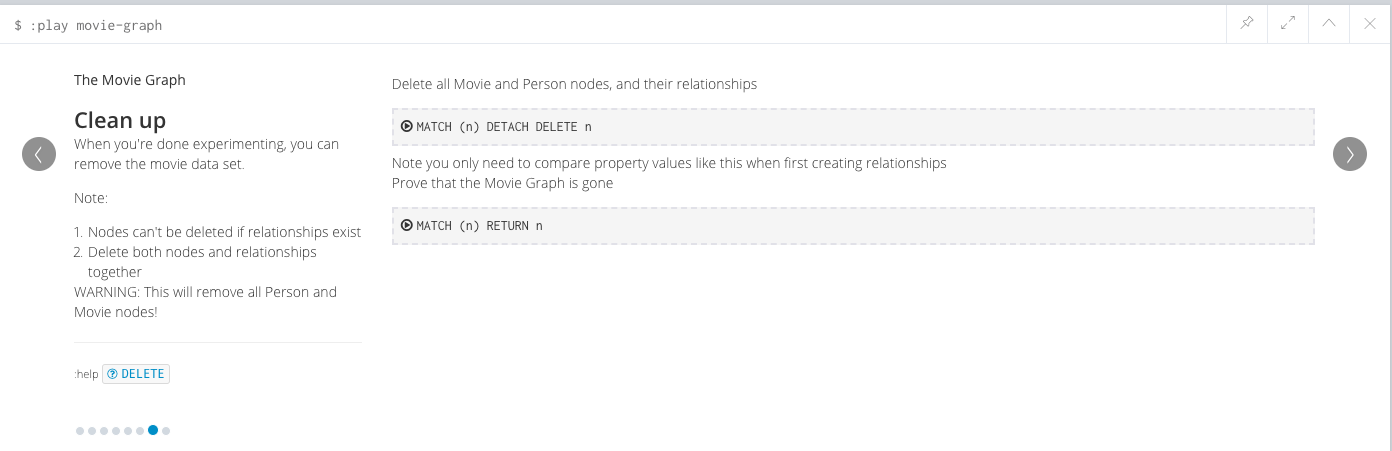
**MATCH (neo:Database {name:"Neo4j"}) MATCH (scott:Person {name:"Scott"}) CREATE (scott)-[:FRIEND]->(:Person:Expert {name:"Andrew"})-[:WORKED\_WITH]->(neo)**



1. **Find someone in your network who can help you learn Neo4j:**

**MATCH (you {name:"Anya"}) MATCH (expert)-[:WORKED\_WITH]->(db:Database {name:"Neo4j"}) MATCH path = shortestPath( (you)-[:FRIEND\*..5]-(expert) ) RETURN db,expert,path**

1. **Perform clean up:**

****

**Part V – Working with Northwind (from RDBMS to Neo4j)**

1. **Execute the following to start the demonstration to create Northwind graph:**

**:play northwind-graph**

Go through all of the commands to create Northwind graph, note how **RDBMS relationships** are converted into **Nodes and Relationship Types**

**SQL TO Cypher Examples**

1. **Select all records from Product table**

SELECT p.\*

FROM product as p;

**Same as matching all nodes with the label :Product and returning them**

**MATCH** (p:Product)

**RETURN p;**

1. **Select ProductName and UnitPrice, ordering by price and returning only the 10 most expansive items:**

SELECT p.ProductName, p.UnitPrice

FROM products as p

ORDER BY p.UnitPrice DESC

LIMIT 10;

**In Cypher:**

**MATCH** (p:Product)

**RETURN p.productName, p.unitPrice**

**ORDER BY p.unitPrice DESC**

**LIMIT 10;**

1. **Select ProductName, UnitPrice with the name ‘Chocolate’**

SELECT p.ProductName, p.UnitPrice

FROM products AS p

WHERE p.ProductName = 'Chocolade';

**In Cypher:**

**MATCH** (p:Product)

**WHERE p.productName = “Chocolade”**

**RETURN p.productName, p.unitPrice;**

**Shortcut if you wanted to match for a labeled node with a specific attribute:**

**MATCH** (p:Product {productName: “Chocolade”})

**RETURN p.productName, p.unitPrice;**

1. **Select productName, unitPrice with product name Chocolade and Chai**

SELECT p.ProductName, p.UnitPrice

FROM products as p

WHERE p.ProductName IN ('Chocolade','Chai');

**In Cypher:**

**MATCH** (p:Product)

**WHERE p.productName IN [ ‘Chocolade’, ‘Chai’]**

**RETURN p.productName, p.unitPrice;**

1. **Select productName, unitPrice with product names starting with C and price > 100**

SELECT p.ProductName, p.UnitPrice

FROM products AS p

WHERE p.ProductName LIKE 'C%' AND p.UnitPrice > 100;

**In Cypher (**LIKE operator is replaced with STARTS WITH)**:**

**MATCH** (p:Product)

**WHERE p.productName STARTS WITH “C” and p.unitPrice > 100**

**RETURN p.productName, p.unitPrice;**